Segment Tree

Bruce Nan

Segment Tree Definition

- Introduced by J. L. Bentley in 1977
- Data structure designed to handle intervals on the real line
- Intervals end points belong to a fixed set of abscissas
- Abscissas can be normalized to range $[1,N]$ without loss of generality by using a lookup table
- Given an interval $[l,r]$, the segment tree $T(l,r)$ is a rooted binary tree defined recursively

Every node v is characterized by two parameters

$B[v]$: beginning of node's world (left end)

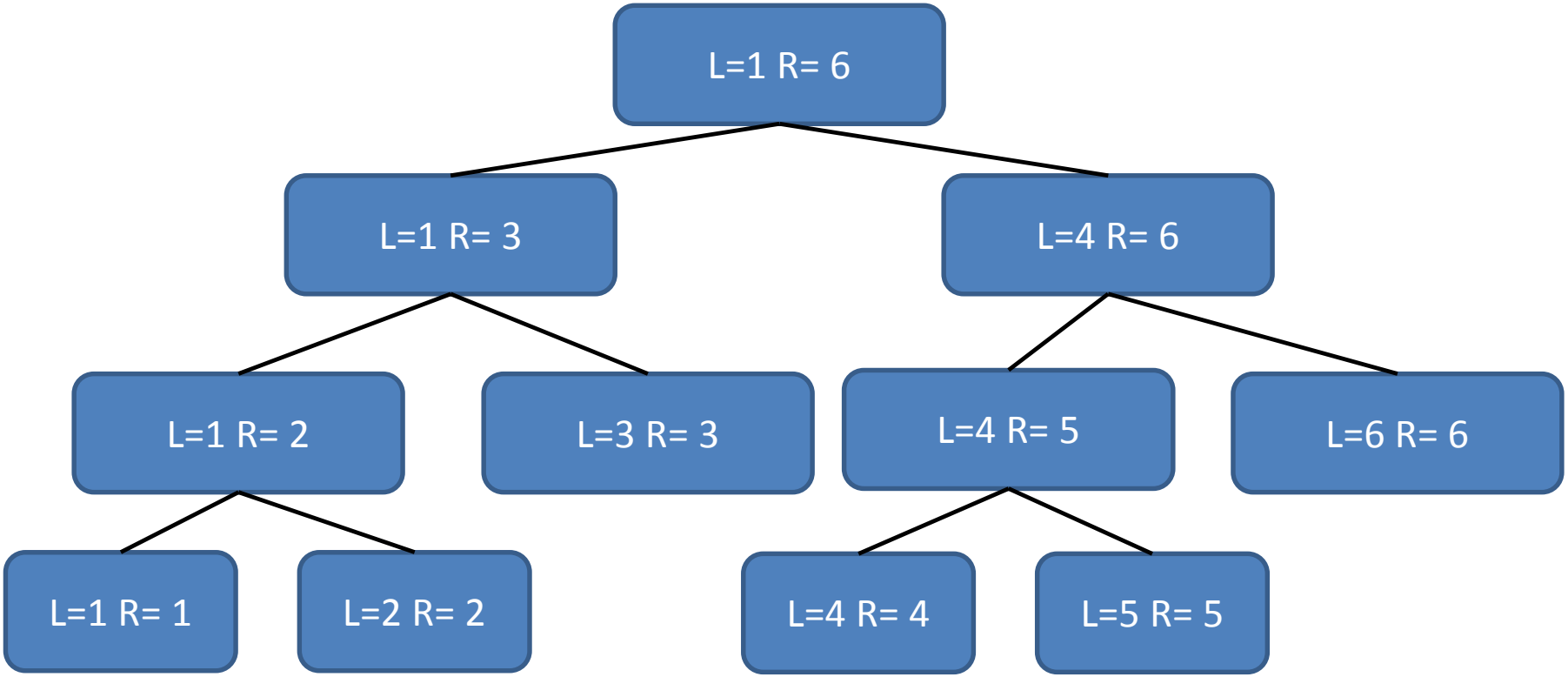
$E[v]$: end of node's world (right end)

If $r-l>1$ a midpoint $\lfloor (B[v]+E[v])/2 \rfloor$ is defined and two sub-trees dividing v's world into two halves are rooted at v:

$LSON[v]$ is the root of a left sub-tree $T\left(l, \lfloor (B[v]+E[v])/2 \rfloor\right)$

$RSON[v]$ is the root of a right sub-tree $T\left(\lfloor (B[v]+E[v])/2 \rfloor, r\right)$

Segment Tree



Features

- Segment tree is a balanced tree. Height <= logN
- Given a segment with length L, it can be the union of 2*logL segments
- Given any two nodes, they are either inclusive or no overlap
- Given a leaf node p, all nodes in the path from root to p all include p

Structure

```
struct node {
        int l, r;
        //data fields
        int Min;
}

Class node {
        int l, r;
        //data fields
        int Min;
}
```

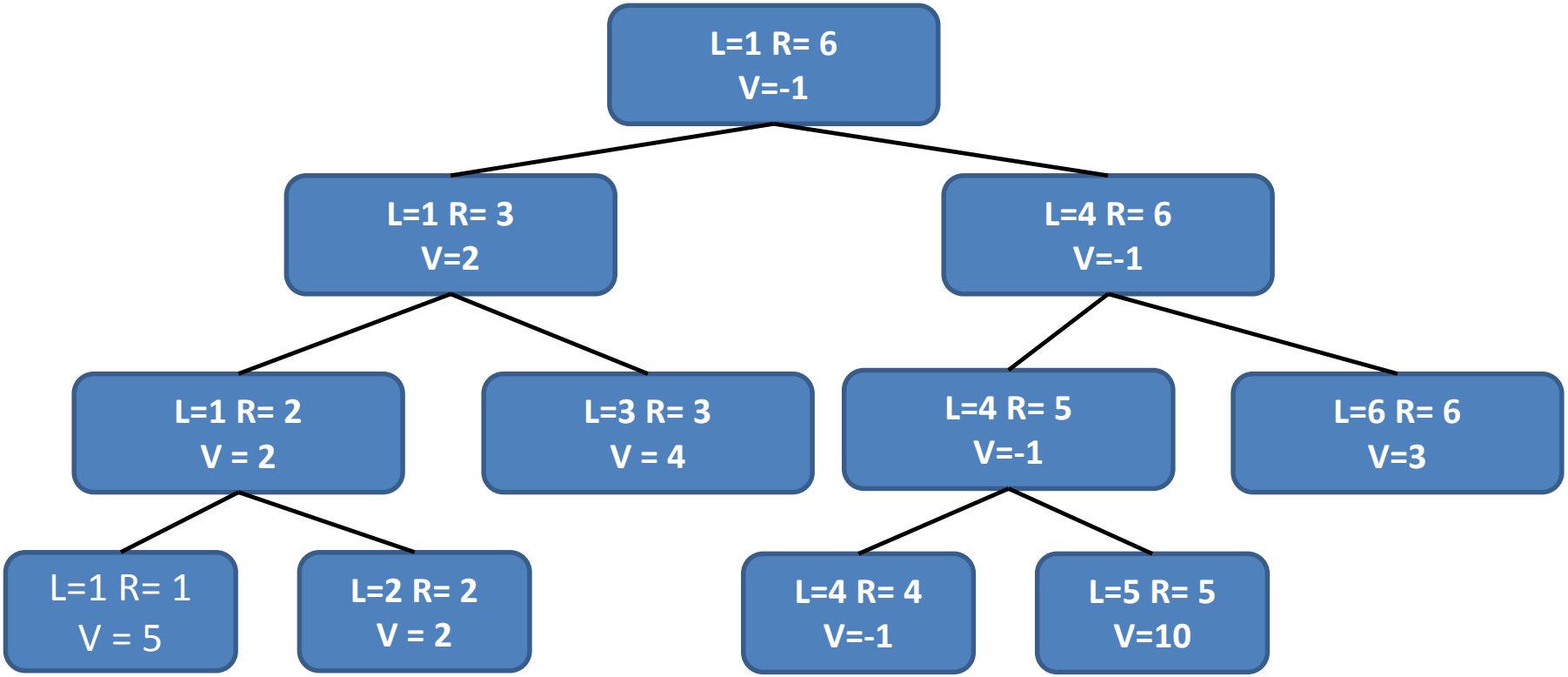
Operations

- Build segment tree
- Update
- Range Minimum Query (RMQ)

Build a segment tree

```
void build(int l, int r, int num){
   seg[num].left = l;
   seg[num].right = r;
   if(l == r){ seg[num].v=a[l]; return;}
   int mid = (l + r) / 2;
   build(l, mid, 2 * num);
   build(mid+1, r, 2 * num + 1);
   seg[num].v=min(seg[2*num].v,
   seg[2*num+1].v);
}
```

Segment Tree with Min Value

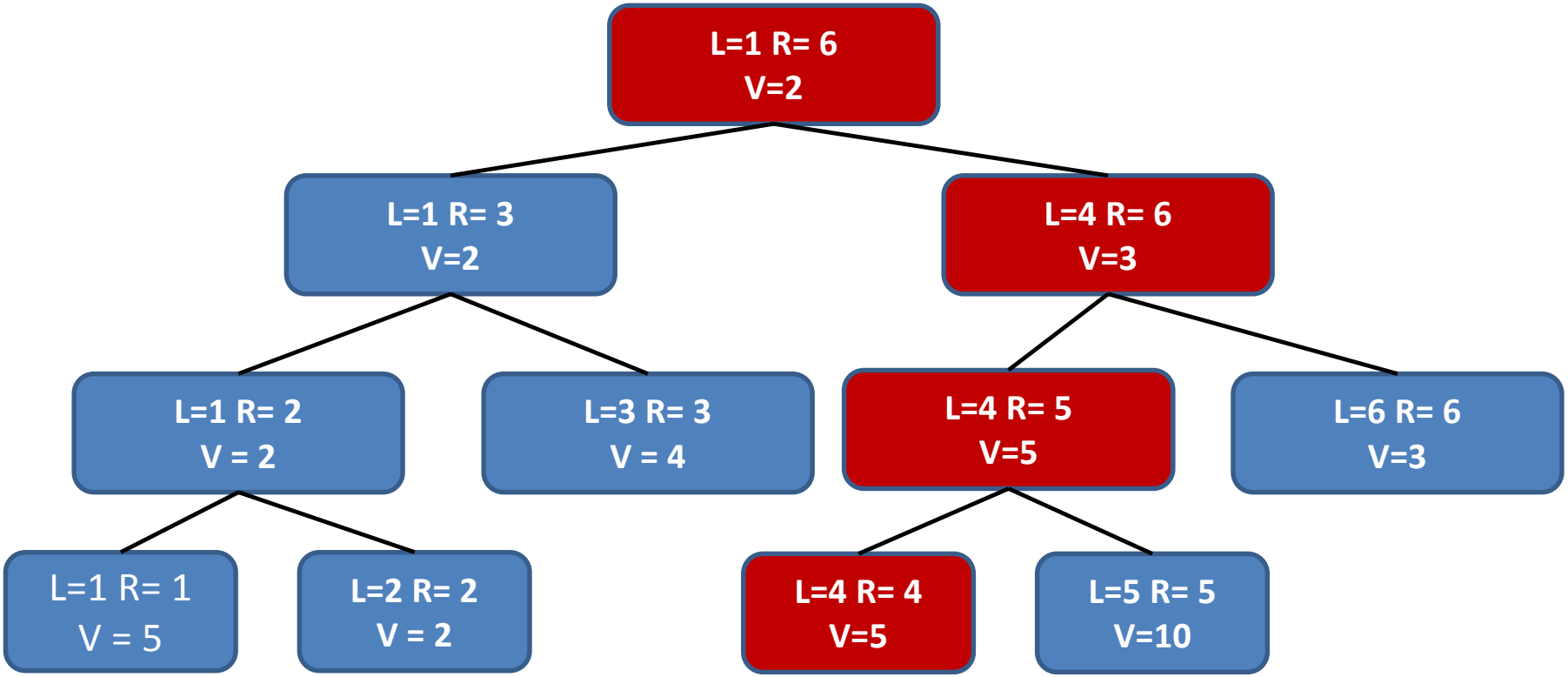


index	1	2	3	4	5	6
a	5	2	4	-1	10	3

Update a single leaf node

```
void update(int pos, int val, int num){
   if (seg[num].left == pos && seg[num].right == pos){
      seg[num].val = val; return;
   }
   Int mid = (seg[num].left + seg[num].right)/2;
   if (pos <= mid)
      update(pos, val, 2 * num);
   else
      update(pos, val, 2 * num +1);
   seg[num].val=min(seg[2*num].val, seg[2*num+1].val);
}
```

Update a[4] = 5



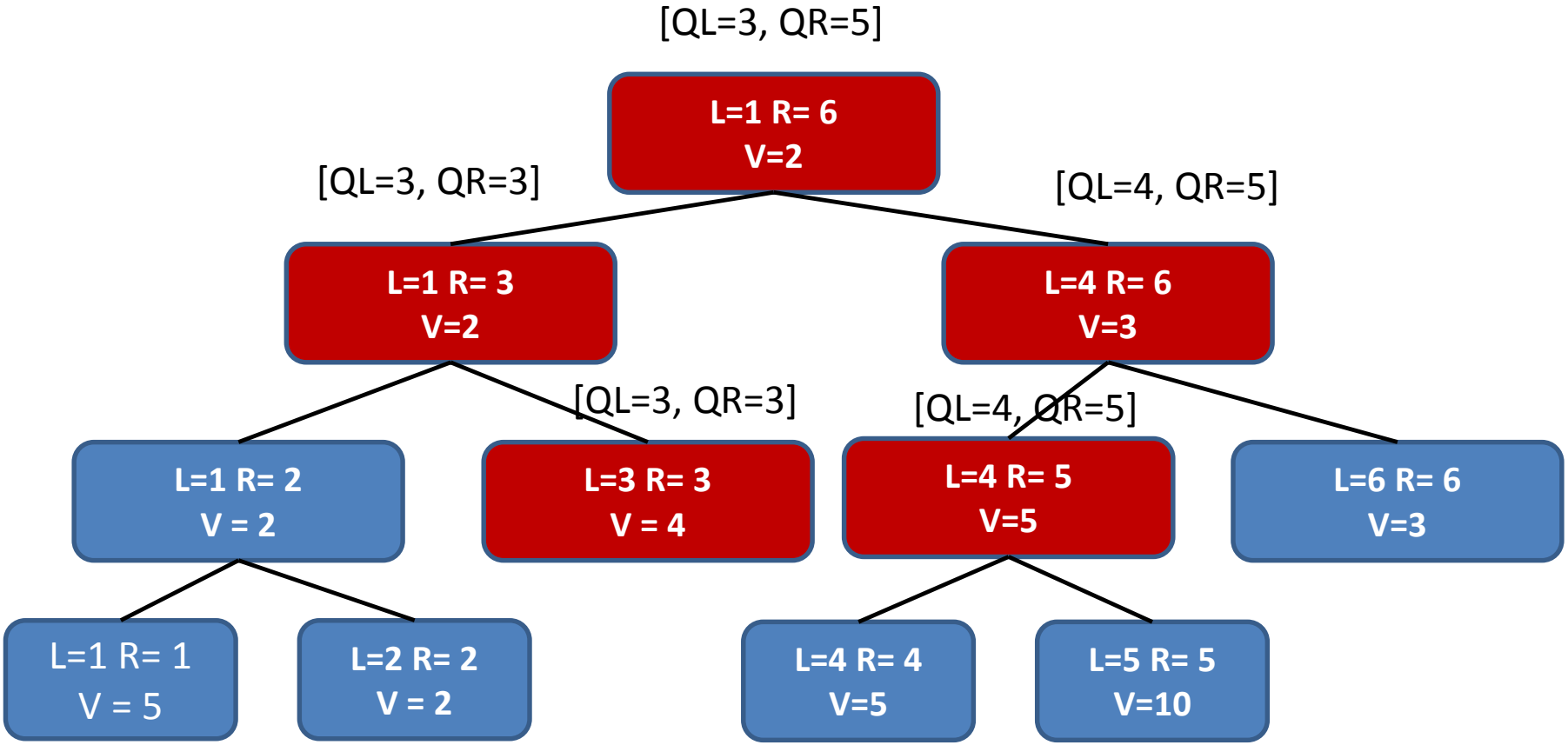
index	1	2	3	4	5	6
a	5	2	4	5	10	3

Query

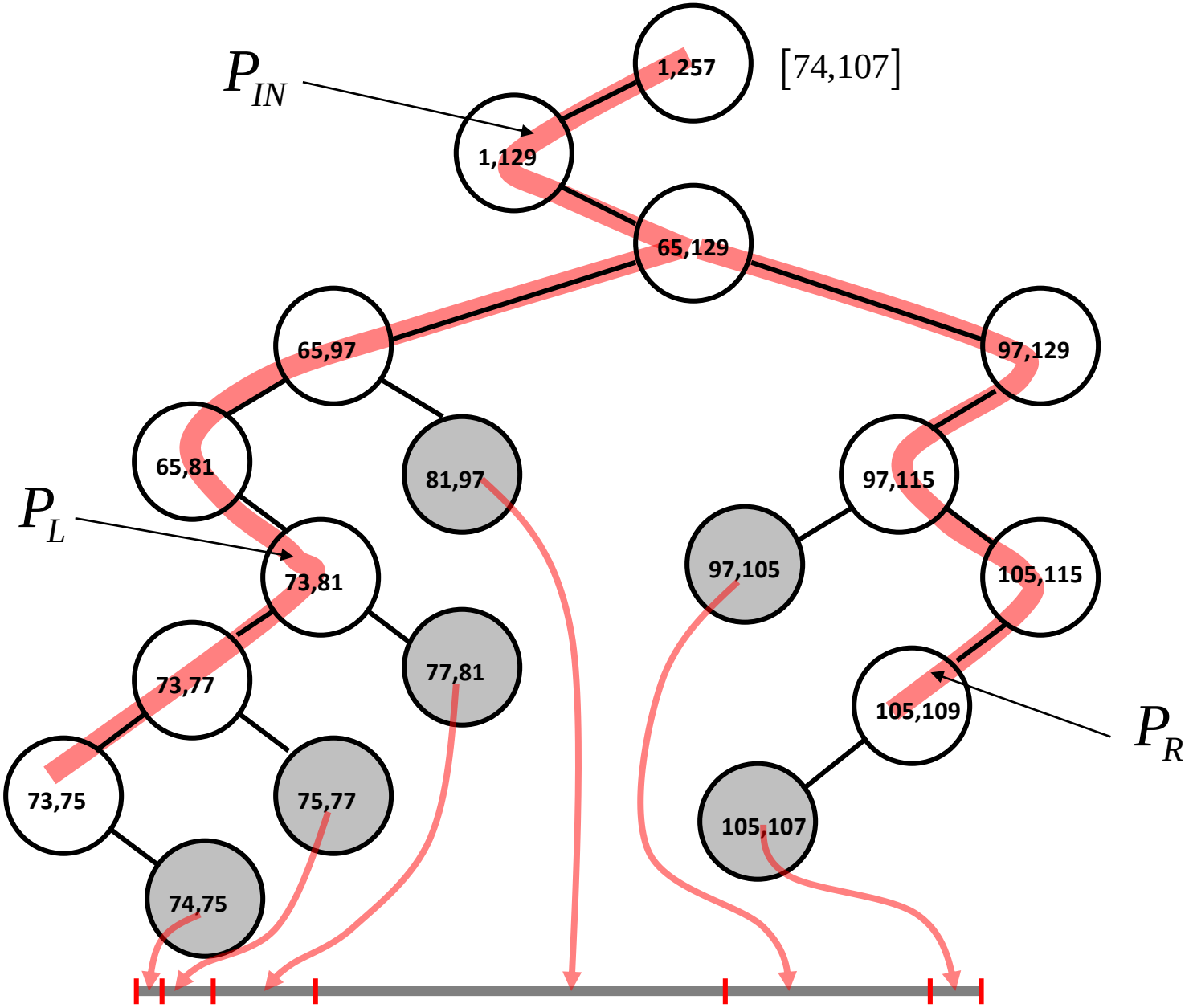
- Range minimal query

```
int Query(int l, int r, int num){
  if(seg [num].l==l && seg [num].r == r)
     return seg[num].val;
  int mid = (seg[num].left+seg[num].right)/2;
  if (r <= mid)
     return Query(l, r, 2 * num);
  else if (l > mid)
     return Query(l, r, 2 * num + 1);
  else return min(Query(l, mid, 2*num),
Query(mid+1, r, 2*num+1);
}
```

Query Min in Range [L=3, R=5]



index	1	2	3	4	5	6
a	5	2	4	5	10	3


P_{IN}
1,257
[74,107]
1,129
65,129
65,97
97,129
65,81
81,97
97,115
P_L
73,81
97,105
105,115
73,77
77,81
105,109
P_R
73,75
75,77
105,107
74,75